Practicum Binaire Zoekbomen

Stijn Lievens

Hogeschool Gent
2018–2019

Binaire Zoekbomen

In dit practicum wordt de implementatie (in Java) van een binaire zoekboom vervolledigd. Je krijgt Java-code voor een binaire zoekboom `BinZoekBoom`. Een aantal methoden hiervan zijn niet geïmplementeerd en gooien een `UnsupportedOperationException`. De bedoeling is dat je deze methoden implementeert. Tracht in eerste instantie alle methodes op een recursieve manier te implementeren.

Je kan controleren of je methodes correct zijn door de gegeven `Junit`-testen uit te voeren. Wanneer alle testen slagen heb je waarschijnlijk alle methodes correct geïmplementeerd.

Uitleg bij de Code

De binaire zoekboom die we implementeren slaat zowel *sleutels* als bijhorende *waarden* op. De sleutels moeten met elkaar vergeleken kunnen worden, vandaar de volgende declaratie van de klasse:

```
public class BinZoekBoom<K extends Comparable<K>, V> {
```

Om het vergelijken van de sleutels gemakkelijker te laten verlopen is er een hulpmethode `compare` voorzien:

```
private Result compare(K k1, K k2) {
  int cmp = k1.compareTo(k2);
  return cmp < 0 ? Result.KLEINER :
         (cmp > 0 ? Result.GROTER : Result.GELIJK);
}
```

Deze methode geeft `Result.KLEINER` wanneer de eerste sleutel kleiner is dan de tweede, `Result.GROTER` wanneer de eerste sleutel groter is dan de tweede of `Result.GELIJK` wanneer beide sleutels gelijk zijn. Wanneer men dus wil vergelijken hoe twee sleutels zich verhouden kan men de volgende code gebruiken:

```
Result cmp = compare(k1, k2);
if (cmp == Result.KLEINER) {
// code wanneer k1 < k2
}
else if (cmp == Result.GROTER) {
// code wanneer k1 > k2
} else {
// code wanneer k1 == k2
}
```

Opgave

1. Start met de implementatie van de methode `voegToe(Top top, K sleutel, V waarde)` die een item toevoegt aan de binaire zoekboom met als wortel `top` én de referentie naar de wortel van de (nieuwe) zoekboom retourneert.

   Het eindgeval van de recursie wordt bereikt wanneer `top` gelijk is aan `null`. In dit geval moet een nieuwe `Top` worden aangemaakt waarvan de referentie wordt geretourneerd door deze methode.

   In het andere geval moet het item worden toegevoegd aan ofwel de linker- of rechterdeelboom; al naargelang het geval moet dus de linker- of rechterwijzer worden aangepast. Nadat dit gebeurd is wordt de top zelf geretourneerd.

   Lees goed de uitleg in de code m.b.t. wat er moet gebeuren indien de sleutel reeds bestaat.

2. Implementeer de methode `getWaarde`; deze methode haalt de waarde op geassocieerd met een bepaalde sleutel. Deze methode is eenvoudiger dan de voorgaande aangezien de boom niet moet worden aangepast. Men kan op eenvoudige recursieve wijze op zoek gaan naar de juiste top in de boom.

3. Implementeer de methode `grootte` die het aantal toppen van de zoekboom berekent. Het aantal toppen van een binaire (zoek)boom is nul wanneer de boom leeg is; in het andere geval is het aantal toppen één plus het aantal toppen in zowel linker- als rechterdeelboom.

4. Implementeer de methode `hoogte` die de hoogte van de binaire zoekboom berekent. De hoogte van een lege binaire (zoek)boom is gelijk aan -1; de hoogte van een niet-lege binaire boom is één meer dan het maximum van de hoogtes van zijn twee deelbomen.

5. Implementeer de methode `getKleinste` om de top met de kleinste sleutel (in een deelboom) te lokaliseren. Wanneer de linkerdeelboom leeg is dan bevat de huidige top het kleinste element en kan deze top worden geretourneerd; in het andere geval behoort het kleinste element tot de linkerdeelboom.

6. Implementeer de methode `sleutels` die een gesorteerde lijst teruggeeft van alle sleutels in de binaire zoekboom. Deze lijst kan bekomen worden door de boom te doorlopen in inorde waarbij de code die wordt uitgevoerd wanneer een top effectief wordt bezocht niets anders is dan het achteraan toevoegen aan de lijst van de sleutel van die top.

7. Zoals gezien in de theorie is de verwijder-methode een relatief ingewikkelde methode. Implementeer eerst de methode `verwijderKleinste` die de top met de kleinste sleutel uit een binaire (deel)zoekboom verwijdert. Implementeer vervolgens de methode `verwijder`. Om het algemene geval (de te verwijderen top heeft twee kinderen) te implementeren maak je gebruik van de methode `verwijderKleinste`.

8. Er is nog een methode `floor` voorzien in de gegeven Java-code. Deze methode moet de volgende functionaliteit hebben: voor een gegeven sleutel k moet deze methode de grootste sleutelwaarde k' teruggeven zodanig dat $k' \leq k$. We kunnen met deze methode dus *benaderend* gaan zoeken in een binaire zoekboom. *Bedenk* eerst hoe je deze methode gaat implementeren. Bekijk voor enkele voorbeelden waar/hoe je de gevraagde waarde kan vinden. Implementeer vervolgens de methode `floor`.

9. Schrijf een tweede methode `sleutels` die een onder- en bovengrens als parameters heeft. Deze methode construeert een gesorteerde lijst van sleutels tussen de twee parameters. Je implementatie mag slechts dié delen van de boom bezoeken waar er zich eventueel sleutels bevinden die aan de voorwaarden voldoen.